```
MMM         MMM  TTTTTTTTTTTTTTT  HHH         HHH  RRRRRRRRRRR     TTTTTTTTTTTTTTT  LLL
MMM         MMM  TTTTTTTTTTTTTTT  HHH         HHH  RRRRRRRRRRR     TTTTTTTTTTTTTTT  LLL
MMM         MMM  TTTTTTTTTTTTTTT  HHH         HHH  RRRRRRRRRRR     TTTTTTTTTTTTTTT  LLL
MMMMMM   MMMMMM        TTT        HHH         HHH  RRR         RRR         TTT        LLL
MMMMMM   MMMMMM        TTT        HHH         HHH  RRR         RRR         TTT        LLL
MMMMMM   MMMMMM        TTT        HHH         HHH  RRR         RRR         TTT        LLL
MMM   MMM   MMM        TTT        HHH         HHH  RRR         RRR         TTT        LLL
MMM   MMM   MMM        TTT        HHH         HHH  RRR         RRR         TTT        LLL
MMM   MMM   MMM        TTT        HHH         HHH  RRR         RRR         TTT        LLL
MMM         MMM        TTT        HHHHHHHHHHHHHHH  RRRRRRRRRRR           TTT        LLL
MMM         MMM        TTT        HHHHHHHHHHHHHHH  RRRRRRRRRRR           TTT        LLL
MMM         MMM        TTT        HHHHHHHHHHHHHHH  RRRRRRRRRRR           TTT        LLL
MMM         MMM        TTT        HHH         HHH  RRR   RRR             TTT        LLL
MMM         MMM        TTT        HHH         HHH  RRR   RRR             TTT        LLL
MMM         MMM        TTT        HHH         HHH  RRR   RRR             TTT        LLL
MMM         MMM        TTT        HHH         HHH  RRR      RRR          TTT        LLL
MMM         MMM        TTT        HHH         HHH  RRR      RRR          TTT        LLL
MMM         MMM        TTT        HHH         HHH  RRR      RRR          TTT        LLL
MMM         MMM        TTT        HHH         HHH  RRR         RRR       TTT        LLLLLLLLLLLLLLL
MMM         MMM        TTT        HHH         HHH  RRR         RRR       TTT        LLLLLLLLLLLLLLL
MMM         MMM        TTT        HHH         HHH  RRR         RRR       TTT        LLLLLLLLLLLLLLL
```

```
UU      UU  VV      VV  XX      XX    SSSSSSSS    QQQQQQ    RRRRRRRR    TTTTTTTTTT
UU      UU  VV      VV  XX      XX    SSSSSSSS    QQQQQQ    RRRRRRRR    TTTTTTTTTT
UU      UU  VV      VV  XX      XX  SS          QQ      QQ  RR      RR      TT
UU      UU  VV      VV  XX      XX  SS          QQ      QQ  RR      RR      TT
UU      UU  VV      VV    XX  XX    SS          QQ      QQ  RR      RR      TT
UU      UU  VV      VV    XX  XX    SS          QQ      QQ  RR      RR      TT
UU      UU  VV      VV      XX        SSSSSS    QQ      QQ  RRRRRRRR        TT
UU      UU  VV      VV      XX        SSSSSS    QQ      QQ  RRRRRRRR        TT
UU      UU  VV      VV    XX  XX            SS  QQ  QQ  QQ  RR  RR          TT
UU      UU  VV      VV    XX  XX            SS  QQ  QQ  QQ  RR  RR          TT
UU      UU    VV  VV    XX      XX          SS  QQ    QQ    RR    RR        TT          ....
UU      UU    VV  VV    XX      XX          SS  QQ    QQ    RR    RR        TT          ....
UUUUUUUUUU      VV      XX      XX  SSSSSSSS      QQQQ  QQ  RR      RR      TT          ....
UUUUUUUUUU      VV      XX      XX  SSSSSSSS      QQQQ  QQ  RR      RR      TT          ....


LL            IIIIII      SSSSSSSS
LL            IIIIII      SSSSSSSS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II        SSSSSS
LL              II        SSSSSS
LL              II              SS
LL              II              SS
LL              II              SS
LL              II              SS
LLLLLLLLLL    IIIIII    SSSSSSSS
LLLLLLLLLL    IIIIII    SSSSSSSS
```

Table of contents

(2)	55	HISTORY ; Detailed Current Edit History
(3)	86	DECLARATIONS ; Declarative Part of Module
(4)	126	MTH$SQRT - Standard Single Precision Floating SQRT
(5)	208	MTH$SQRT_R3 - JSB SQRT routine

```
0000          .TITLE  UVX$SQRT          ; Floating Point Square Root routine
0000                                    ; (SQRT)
0000          .IDENT /1-016/            ; File: MTHSQRT.MAR     EDIT JCW1016
0000  ;
0000  ;*******************************************************************************
0000  ;*                                                                             *
0000  ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                    *
0000  ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                     *
0000  ;*  ALL RIGHTS RESERVED.                                                       *
0000  ;*                                                                             *
0000  ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED       *
0000  ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE      *
0000  ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER       *
0000  ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY       *
0000  ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY       *
0000  ;*  TRANSFERRED.                                                               *
0000  ;*                                                                             *
0000  ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE       *
0000  ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT      *
0000  ;*  CORPORATION.                                                               *
0000  ;*                                                                             *
0000  ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS       *
0000  ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                    *
0000  ;*                                                                             *
0000  ;*                                                                             *
0000  ;*******************************************************************************
0000  ;
0000  ;
0000  ; FACILITY: MATH LIBRARY
0000  ;++
0000  ; ABSTRACT:
0000  ;
0000  ; MTH$SQRT is a function  which  returns the floating point square root
0000  ; of its single precision floating point argument. The call is standard
0000  ; call-by-reference.
0000  ; MTH$SQRT_R3 is a special routine which is the same as MTH$SQRT except
0000  ; a faster non-standard JSB call is used with the argument in R0 and no
0000  ; registers are saved.
0000  ;
0000  ;--
0000  ;
0000  ; VERSION: 01
0000  ;
0000  ; HISTORY:
0000  ; AUTHOR:
0000  ;       Peter Yuo, 15-Oct-76: Version 01
0000  ;
0000  ; MODIFIED BY:
0000  ;
0000  ; 01-1  Peter Yuo, 22-May-77
0000  ; 01-2 Peter Yuo, 31-May-77
0000  ; 1-016 Jeffrey C. Wiener 22-Feb-83
0000  ;
```

```
0000      .SBTTL  HISTORY ; Detailed Current Edit History
0000
0000
0000  ; ALGORITHMIC DIFFERENCES FROM FP-11/C ROUTINE: none
0000  ;
0000  ; Edit History for Version 01 of MTH$SQRT
0000  ;
0000  ; 01-1  Code saving after code review
0000  ; 01-2  ROTL shift in garbage into highest bit. Use ASHL instead.
0000  ;        ADDL instruction after ADJUST has been changed into ADDW to prevent
0000  ;        overflow if R1<31:16> = FFFF and R0<31:16> = FFFF
0000  ; 01-3  Finish error handling 10-June-1977
0000  ; 01-5  MTH$$ERROR changed to MTH$$SIGNAL.
0000  ;        MTH$_... changed to MTH__....
0000  ;        Changed error handling mechanism. Put error result in R0 before
0000  ;        calling MTH$$SIGNAL in order to allow user modify error result.
0000  ; 01-6  Return -0.0 on negative arg.  TNH 20-Dec-77
0000  ; 01-7  Edit in Rich Lary's code bums.  JSB routine is now _R3.  JMT 19-Jan-78
0000  ; 01-9  Move .ENTRY symbol to module header.  TNH 14-Aug-78
0000  ; 1-010 - Put version number in standard format: three digit edit
0000  ;          numbers.  Also, update the copyright notice.  JBS 16-NOV-78
0000  ; 1-011 - Change MTH__SQUROONEG to MTH$K_SQUROONEG.  JBS 07-DEC-78
0000  ; 1-012 - Add "_" to the PSECT directive.  JBS 22-DEC-78
0000  ; 1-013 - Declare externals.  SBL 17-May-1979
0000  ; 1-014 - Move MTH$SQRT_R2 to separate module (MTHSQRTR2.MAR) and
0000  ;          replace with MTH$SQRT_R3.  JAW 26-Sep-1979.
0000  ; 1-015 - Changed W^ to G^ in call to MTH$$SIGNAL RNH 09-Sept-1981
0000  ; 1-016 - Changed all references to D_floating to G_floating. This was
0000  ;          done because G_floating, not D_floating, is to be used as
0000  ;          back-up to F_floating. JCW 22-FEB-83
```

```
                0000          .SBTTL  DECLARATIONS    ; Declarative Part of Module
                0000
                0000  ;
                0000  ; INCLUDE FILES:
                0000  ;
                0000
                0000  ;
                0000  ; EXTERNAL SYMBOLS:
                0000  ;
                0000          .DSABL  GBL
                0000          .EXTRN  MTH$K_SQUROONEG
                0000          .EXTRN  MTH$$SIGNAL
                0000
                0000  ;
                0000  ; EQUATED SYMBOLS:
                0000  ;
                0000
0000400C        0000          ACMASK = ^M<IV, R2, R3>         ; register save mask and IV enable
                0000  ; MACROS:       none
                0000  ;
                0000  ; PSECT DECLARATIONS:
                0000
00000000        0000          .PSECT  _MTH$CODE       PIC,SHR,LONG,EXE,NOWRT
                0000                                          ; program section for math routines
                0000  ;
                0000  ; OWN STORAGE:  none
                0000  ;
                0000  ; CONSTANTS:
                0000
                0000  ;
                0000  ; Constants A and B chosen for k = odd
                0000  ;
13CD5FD4        0000          LF_ODD_A_E63    =       ^X13CD5FD4
3C4A2018        0000          LF_ODD_B_EM63   =       ^X3C4A2018
                0000  ;
                0000  ; Constants A and B chosen for k = even
                0000  ;
F61A4015        0000          LF_EVEN_A       =       ^XF61A4015
4B231FD7        0000          LF_EVEN_B_EM64  =       ^X4B231FD7
```

```
                    .SBTTL  MTH$SQRT  - Standard Single Precision Floating SQRT


;++
; FUNCTIONAL DESCRIPTION:
;
; SQRT  - single precision floating point function
;
; SQRT(X) is computed using the following approximation technique:
;
;       If X <= 0 , error.  Let X = |X|.
;
;       Let X = 2**K * F where F is the fractional part.
;
;       If K = even, X = 2**(2P) * F,
;               SQRT(X) = 2**P * SQRT(F), 1/2 =< F < 1
;
;       If K = odd, X = 2**(2P+1) * F = 2**(2P+2) * (F/2),
;               SQRT(X) = 2**(P+1) * SQRT(F/2), 1/4 =< F/2 < 1/2.
;
;       Let F' = A*F + B,
;                       A = 0.453730314(octal),
;                       B = 0.327226214(octal), for K = even.
;               = A*(F/2) + B,
;                       A = 0.650117146(octal),
;                       B = 0.230170444(octal), for K = odd.
;       and
;           K' = P,         for K = even
;              = P + 1      for K = odd.
;
;       Let Y0 = 2**K' * F' as a staight line approximation wthin the
;       given interval using coefficients A and B which minimize the
;       absolute error at the midpoint and endpoint.
;
;       Starting with Y0, two Newton-Raphson iterations are performed.
;
;       Y[n+1] = (1/2) * ( Y[n] + X/Y[n])
;
;       The relative error is < 10**-8.
;
; CALLING SEQUENCE:
;
;       sqrt.wf.v = MTH$SQRT(x.rf.r)
;
; INPUT PARAMETERS:

        LONG = 4                        ; define longword multiplier
        x = 1 * LONG                    ; Contents of x is the argument

; IMPLICIT INPUTS:      none
;
; OUTPUT PARAMETERS:
;
;       VALUE:  floating square root of the argument
;
; IMPLICIT OUTPUTS:     none
;
```

```
                                0000   ; COMPLETION CODES:     none
                                0000   ;
                                0000   ; SIDE EFFECTS:
                                0000   ;
                                0000   ; Signals: MTH$_SQUROONEG if X < 0.0 with reserved operand in R0 (copied to
                                0000   ; the signal mechanism vector CHF$L_MCH_R0/R1 by LIB$SIGNAL).
                                0000   ; Associated message is: "SQUARE ROOT OF NEGATIVE VALUE". Result is reserved
                                0000   ; operand -0.0 unless a user supplied (or any) error handler changes CHF$L_MCH_R0/R1
                                0000   ;
                                0000   ; NOTE: This procedure disables floating point underflow, enables integer
                                0000   ; overflow, causes no floating overflow or other arithmetic traps, and
                                0000   ; preserves enables across the call.
                                0000   ;
                                0000   ;---
                                0000
                                0000
                          400C  0000          .ENTRY  MTH$SQRT, ACMASK          ; standard call-by-reference entry
                                0002                                            ; disable DV (and FU), enable IV
                                0002          MTH$FLAG_JACKET                   ; flag that this is a jacket procedure in
                                0002
     6D   00000000'GF     9E    0002          MOVAB   G^MTH$$JACKET_HND, (FP)
                                0009                                            ; set handler address to jacket
                                0009                                            ; handler
                                0009
                                0009                                            ; case of an error in special routine
          50   04 BC     50    0009          MOVF    @x(AP), R0                ; R0 = arg
                 01      10    000D          BSBB    MTH$SQRT_R3               ; call specail SQRT rountine
                         04    000F          RET                               ; return - result in R0
                               0010
```

```
                                0010          .SBTTL  MTH$SQRT_R3  - JSB SQRT routine
                                0010
                                0010  ; JSB SQRT - used by the standard, and directly.
                                0010  ;
                                0010  ; CALLING SEQUENCE:
                                0010  ;       save anything in R0:R2
                                0010  ;       MOVF    ..., R0                 ; input in R0
                                0010  ;       JSB     MTH$SQRT_R3
                                0010  ;       return with result in R0
                                0010  ;
                                0010  ; Note: This routine is written to avoid any integer overflows, floating overflows,
                                0010  ; floating underflows or divide by 0 conditions, whether enabled or not.
                                0010  ;
                                0010  ; REGISTERS USED:
                                0010  ;       R0 - Floating argument then result
                                0010  ;       R1 - X saved for use during iteration
                                0010  ;       R2 - scratch
                                0010
                                0010  MTH$SQRT_R3::                           ; JSB routine for SQRT
          51    50    50        0010          MOVF    R0, R1                  ; test sign of X and save it in R1.
                62    15        0013          BLEQ    ZERO_NEG                ; branch to ZERO_NEG if X =< 0
                                0015  ;
                                0015  ; X > 0
                                0015  ;
                                0015  POS:
          52    50    3C        0015          MOVZWL  R0, R2                  ; isolate low 16 bits (sign,exp,>fract) in R
                52    94        0018          CLRB    R2                      ; R2 now has sign and left 7 exp bits
          50    52    AA        001A          BICW    R2, R0                  ; clear sign and left 7 exp bits
                50    95        001D          TSTB    R0                      ; check low bit of exp
                10    18        001F          BGEQ    EVEN                    ; and branch if 1
50    13CD5FD4 8F    44        0021          MULF    #LF_ODD_A_E63, R0       ; add 64 (half of bias) to (exponent-2)
                                0028                                          ; and start approximation calc
50    3C4A2018 8F    40        0028          ADDF    #LF_ODD_B_EM63, R0      ; R0 = (first approx) * 2**-64
                13    11        002F          BRB     ADJUST                  ; go adjust
                                0031
                                0031  EVEN:
     50    2000 8F    A0        0031          ADDW    #^X2000, R0             ; exp is 0 - make it 64 (2**-64) for legalit
50    F61A4015 8F    44        0036          MULF    #LF_EVEN_A, R0
50    4B231FD7 8F    40        003D          ADDF    #LF_EVEN_B_EM64, R0     ; R0 = (first approx) * 2**-64
                                0044  ADJUST:
     52    52    1F    9C        0044          ROTL    #31, R2, R2             ; divide R2 (exp+bias) by 2,
                                0048                                          ; giving (exp/2+64)
          50    52    A0        0048          ADDW    R2, R0                  ; insert exp/2 in first approx and
                                004B                                          ; re-bias it.
                                004B
                                004B  ; first iteration - single precision is sufficient
                                004B  ;
     52    51    50    47        004B          DIVF3   R0, R1, R2              ; R2 = X/Y0
          50    52    40        004F          ADDF    R2, R0                  ; R0 = Y0 + X/Y0
     50    0080 8F    A2        0052          SUBW    #^X80, R0               ; R0 = Y1 = (1/2)(Y0 + X/Y0)
                                0057                                          ; no overflow possible
                                0057
                                0057  ; second iteration, do in double precision to get truncated( rather than
                                0057  ; rounded) result.
                                0057  ;
                                0057  ;;;     CLRL    R2                      ; lower part (X) = 0
                                0057  ;;;     DIVD    R0, R1                  ; divide Y1 into X with low-order
```

```
                        0057                                                ; 32 bits of Y1 garbage.  This doesn't
                        0057                                                ; effect accuracy, since Y1 inacurate
                        0057                                                ; anyway.
                        0057  ;;;     Using DIVG instead we have
                        0057  ;;;     DIVG2   R0 ,R1                        ; divide Y1 into X with low-order 53-24=
                        0057                                                ; 29 bits of Y1 garbage.  This doesn't
                        0057                                                ; effect accuracy, since Y1 inacurate
                        0057                                                ; anyway.
                        0057
          52   51 99FD  0057          CVTFG   R1, R2                        ; convert and copy X into R2/R3
               51   D4  005B          CLRL    R1                            ; clear low part of Y1
          7E   50 99FD  005D          CVTFG   R0, -(SP)
          52   8E 46FD  0061          DIVG2   (SP)+, R2                     ; divide Y1 into X
                        0065  ;
                        0065  ; Need to CVTGF without rounding. To do this clear the rounding bit.
                        0065  ;  The rounding bit is bit 12 of the 3rd word(word 2 starting from 0).
                        0065  ;
     53   1FFF 8F   AA  0065          BICW2   #^X1FFF, R3                   ; Clear the rounding bit so that
          52   52 33FD  006A          CVTGF   R2, R2                        ;  CVTGF truncates instead of rounds.
                        006E  ;
                        006E  ; Back to original code.
                        006E  ;
          50   52   40  006E          ADDF    R2, R0                        ; R0 = Y1 + higher_part(X/Y1)
     50   0080 8F   A2  0071          SUBW    #^X80, R0                     ; R0 = SQRT(X) = (1/2) (Y1 + X/Y1)
                   05  0076  SQRTX:   RSB                                   ; return, R0 = result
                        0077
                        0077  ; X =< 0
                        0077  ;
                        0077  ZERO_NEG:
               FD   13  0077          BEQL    SQRTX                         ; return with R0 = result = 0
               6E   DD  0079          PUSHL   (SP)                          ; return PC from JSB routine
       7E   00'8F   9A  007B          MOVZBL  #MTH$K_SQUROONEG, -(SP)       ; condition value
     50   01   0F   78  007F          ASHL    #15, #1, R0                   ; R0 = result = reserved operand -0.0
                        0083                                                ; R0 goes to signal mechanism vector
                        0083                                                ; (CHF$L_MCH_R0/R1) so error handler
                        0083                                                ; can modify the result.
00000000'GF     02   FB  0083          CALLS   #2, G^MTH$$SIGNAL             ; signal error and use real user's PC
                        008A                                                ; independent of CALL vs JSB
                   05  008A          RSB                                   ; return - R0 restored from CHF$L_MCH_R0/R1
                        008B
                        008B          .END
```

```
ACMASK          = 0000400C
ADJUST            00000044 R     01
EVEN              00000031 R     01
LF_EVEN_A       = F61A4015
LF_EVEN_B_EM64  = 4B231FD7
LF_ODD_A_E63    = 13CD5FD4
LF_ODD_B_EM63   = 3C4A2018
LONG            = 00000004
MTH$$JACKET_HND   ********   X   01
MTH$$SIGNAL       ********   X   00
MTH$K_SQUROONEG   ********   X   00
MTH$SQRT          00000000 RG    01
MTH$SQRT_R3       00000010 RG    01
POS               00000015 R     01
SQRTX             00000076 R     01
X               = 00000004
ZERO_NEG          00000077 R     01
```

+----------+
! Psect synopsis !
+----------+

```
PSECT name                      Allocation          PSECT No.  Attributes
----------                      ----------          ---------  ----------
.  ABS  .                       00000000  (    0.)  00 (  0.)  NOPIC   USR   CON   ABS   LCL NOSHR NOEXE NORD  NOWRT NOVEC BYTE
_MTH$CODE                       0000008B  (  139.)  01 (  1.)    PIC   USR   CON   REL   LCL   SHR   EXE   RD  NOWRT NOVEC LONG
```

+----------+
! Performance indicators !
+----------+

Phase	Page faults	CPU Time	Elapsed Time
Initialization	29	00:00:00.11	00:00:00.68
Command processing	107	00:00:00.64	00:00:02.67
Pass 1	82	00:00:00.81	00:00:03.90
Symbol table sort	0	00:00:00.00	00:00:00.00
Pass 2	68	00:00:00.68	00:00:02.45
Symbol table output	3	00:00:00.03	00:00:00.21
Psect synopsis output	2	00:00:00.02	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	293	00:00:02.29	00:00:09.93

The working set limit was 900 pages.
4312 byies (9 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 17 non-local and 0 local symbols.
365 source lines were read in Pass 1, producing 11 object records in Pass 2.
1 page of virtual memory was used to define 1 macro.

+----------+
! Macro library statistics !
+----------+

Macro library name	Macros defined
_255DUA28:[SYSLIB]STARLET.MLB;2	0

```
0 GETS were required to define 0 macros.

There were no errors, warnings or information messages.

MACRO/ENABLE=SUPPRESSION/DISABLE=(GLOBAL,TRACEBACK)/LIS=LIS$:UVXSQRT/OBJ=OBJ$:UVXSQRT MSRC$:MTHJACKET/UPDATE=(ENH$:MTHJACKET)+MSRC$:
```

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

NCP
MAP

UUXSQRTR2
LIS

UUXSQRT
LIS

NCPDEF
SDL

UUXSINCOS
LIS

NMADEF
SDL

NCP